# Lecture Summary

NET summary

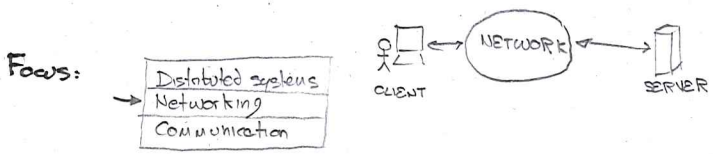
## Table of Contents

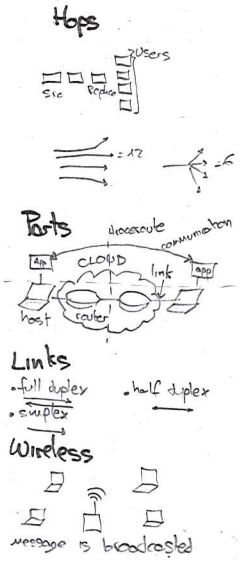

0 Fundamentals ..... 2
1 Physical Layer ..... 3
2 Data Link Layer ..... 4
3 Network Layer ..... 10
4 Transport Layer ..... 17
~~5~~ Session Layer ..... 26
~~6~~ Presentation Layer ..... 26
7 Application Layer ..... 26


PDN TS PA

## Info

There is no claim for completeness. All warranties are disclaimed. Creative Commons Attribution-Noncommercial 3.0 Unported license.








## Study Part

This document is structured according to the OSI model of layers.
Mnemonics: osi7layer.com and reddit.com/r/networking/comments/1n27uw/
Stuff: github: @alex/what-happens-when, blog.cloudflare.com

## 0 Fundamentals

(just that)

**Multiplexing** means sharing, **statistical** multiplexing shares e.g. network bandwidth between users according to the statistics of their demand and is useful because users are mostly idle and their traffic is bursty.

**Metcalfe's Law** states the value of a network of $n$ nodes is proportional to $n^2$ which means large networks are relatively more valuable than small ones.

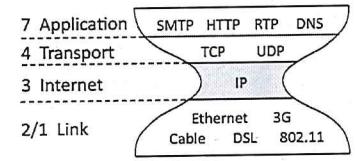
| SCALE | TYPE | EXAMPLE |
|---|---|---|
| VICINITY | PAN (Personal Area Network) | Bluetooth (e.g., headset) |
| BUILDING | LAN (Local Area Network) | WiFi, Ethernet |
| CITY ~~MAN~~ | (Metropolitan Area Network) MAN | Cable, DSL |
| COUNTRY | WAN (Wide Area Network) | Large ISP (internet service provider) |
| PLANET | The Internet (network of all networks) | The Internet! |

other networks — Mobile (2G, 3G, 4G) — telephone / satellite — tor

**Traceroute** is widely used to let hosts peek inside the network by successively proving hops. (→ to find path in network) **Protocols and layering** is the main structuring method used to divide up network functionality. Each instance of a protocol talks virtually to its peer using the protocol. Each instance of a protocol uses only the services of the lower layer. Protocols are horizontal, layers are vertical. Encapsulation is used to effect protocol layering where a lower layer wraps higher layer content adding its own information for message delivery. E.g. [802.11 [IP [TCP [HTTP]]]]. The advantages of layering are: information hiding and reuse and using information hiding to connect different systems. Disadvantages are: it adds overhead and hides information (e.g. whether wired or wireless).

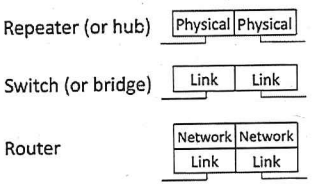
The **OSI 7 Layer Reference model** is listed below and the experience-based internet reference model is pictured.

| | | |
|---|---|---|
| 1. | Physical | sends bits as signals |
| 2. | Data link | provides functions needed by users |
| 3. | Network | converts different data representations |
| 4. | Transport | manages task dialogs |
| ~~5.~~ | Session | provides end-to-end delivery |
| ~~6.~~ | Presentation | sends packets over multiple links |
| 7. | Application | sends frames of information |



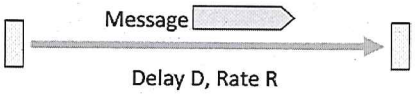

For different layers, different names for units of data are used: see table to the right. For devices, see below:

| LAYER | UNIT OF DATA |
|---|---|
| ~~7~~ APPLICATION | Message |
| 4 TRANSPORT | Segment |
| 3 NETWORK | Packet |
| 2 LINK | Frame |
| 1 PHYSICAL | Bit |



## 1 Physical Layer

*Concerns how signals are used to transfer message bits over a link.*




### Latency

The **latency** $L$ is delay to send a message over a link with rate $R$, length $\ell$ and speed $v = 2/3\,c$ and consists of the transmission delay $T = M/R$ [s] which is the time to put an $M$-bit message on the wire and the propagation delay $D = \ell/v$ being the time for the bits to propagate across the wire. $L = M/R + D$. Example with $M = 1250$ bytes and dialup: $D = 5\,ms, R = 56\,kbps \rightarrow L = 5\,ms + (1250 \cdot 8)/(56 \cdot 10^3)\,s = 184\,ms$ and broadband: $D = 50\,ms, R = 10Mbps \rightarrow L = 50\,ms + (1250 \cdot 8)/(10 \cdot 10^6)\,s = 51\,ms$. A long link or a slow rate means high latency. Additionally, messages take space on the wire and the amount of data in flight is the bandwidth-delay $BD = R \cdot D$ product, measures in bits or messages. Example with $R = 40\,Mbps, D = 50ms \rightarrow BD = 40 \cdot 10^6 \cdot 50 \cdot 10^{-3} bits = 2000Kbit = 250\,KB$.

> Units: B = bytes, b = bits, power of 10 for rates, power of 2 for storage and data size

### Signal Carriers

**Twisted pair wires** are very common and reduce radiated signal or reduce effect of external interference signal. **Coaxial cables** are common as well and have better shielding which increases performance. **Fiber** is very thin and is really high speed over long distances. Sending signals over **wireless** sends the signal in many directions and there might be interference[1].

A signal propagates along the wire and fewer frequencies i.e. less bandwidth degrades the signal. When a signal passes over a wire, the following things happen:

1. The signal is delayed (propagates at ⅔ c)
2. The signal is attenuated
3. Frequencies above a cutoff are highly attenuated
4. Noise is added to the signal (later, causes errors)

When passing through fiber, very little information is lost and when the transmission is over wireless, it travels at the speed of light and interference leads to the notion of spatial reuse of the same frequency. **Wireless multipath** is what happens when the signals bounces off objects and takes multiple paths. This messes up the signal, varies by location and requires sophisticated methods for resolution.

### Modulation

Modulation deals with the problem of how to represent bits in signals. A very simple modulation is where high voltages represent a 1, low voltages a 0 (aka NRZ). **Clock recovery** maps every 4 data bits into 5 code bits without long runs of zeroes (since they are hard to count) and then there are at most 3 zeroes in a row. All these examples are baseband modulation where the signal is sent directly on a wire, and these signals don't propagate well on fiber/wireless where higher frequencies are required. This is done using passband modulation where the carrier is modulated (by changing amplitude, frequency, or phase).

[1] One of the reasons for 5 GHz WiFi.





### Fundamental Limits

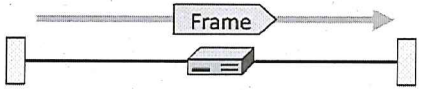
There are limits on how fast information can be sent: Nyquist limit and the Shannon capacity. The bandwidth $B$, singal strength $S$, and noise strength $N$ contribute in different ways: $B$ limits the rate of transitions, $S, N$ limit how many signal levels can be distinguished. According to the Nyquist limit, the maximum symbol rate is $2B$ and at $V$ signal levels (without noise), the maximum bit rate is $R = 2B \log_2 V$ bits/sec. The number of signal levels that can be distinguished depends on $S/N$ which is the signal-to-noise ratio SNR (noise is random). The Shannon capacity limit, the maximum information carrying rate of the channel is $C = B \log_2(1 + S/N)$ bits/sec.

For wired and fiber connections, the SNR is engineered for data i.e. data rate can be fixed. When operating wirelessly, the data rate is adapted to SNR i.e. cannot design for worst case.

**DSL** reuses twisted pairs telephone line, uses passband modulation, separate upstream/downstream bands, the modulation varies both amplitude and phase, and can vary the SNR between 1 and 15 bits per symbol.

## 2 Data Link Layer

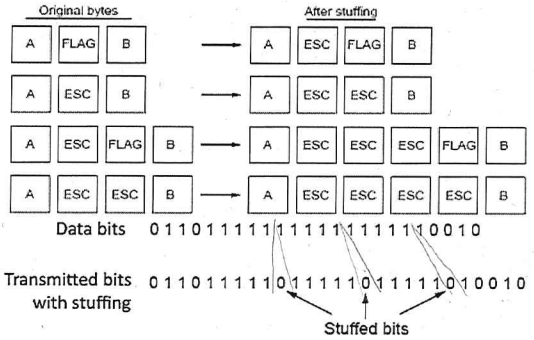
*Concerns how to transfer messages over one or more connected links.*




### Framing

Framing deals with how to transform the continuous stream from the physical layer into "messages" aka frames. A very simple approach is **counting bytes** i.e. length fields. Re-synchronization after a framing error is difficult.

**Byte stuffing** uses a special flag byte value to signal start/end of a frame and replaces the flag inside the frame with an escape code (which then needs to be escaped again). This also works a bit level e.g. six consecutive 1s are a flag.


Original bytes
After stuffing
A FLAG B → A ESC FLAG B
A ESC B → A ESC ESC B
A ESC FLAG B → A ESC ESC ESC FLAG B
A ESC ESC B → A ESC ESC ESC ESC B
Data bits 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
Transmitted bits with stuffing 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0
Stuffed bits

An example protocol using byte stuffing is PPP over SONET

### Error detection and correction

Due to noise, errors happen can be detected and/or corrected (retransmission will be dealt with later). Reliability is a concern across all layers. The approach taken here is to add redundancy i.e. check bits.

Using error codes, the codeword consists of $D$ data and $R$ check bits. The sender computes $R$ based on $D$ and sends $D + R$. The receiver then recomputes $R$ based on $D$ and throws an error if they don't match.

The **Hamming distance** of a code is the minimum distance between any pair of codewords. The distance is defined as the number of bit flips needed to change $D + R_1$ to $D + R_2$. Using Hamming distance on a code of distance $d + 1$ up to $d$ errors will always be *detected* and for a code of distance $2d + 1$ up to $d$ errors can always be *corrected* (by mapping to the closest codeword).

Approaches for error detection:

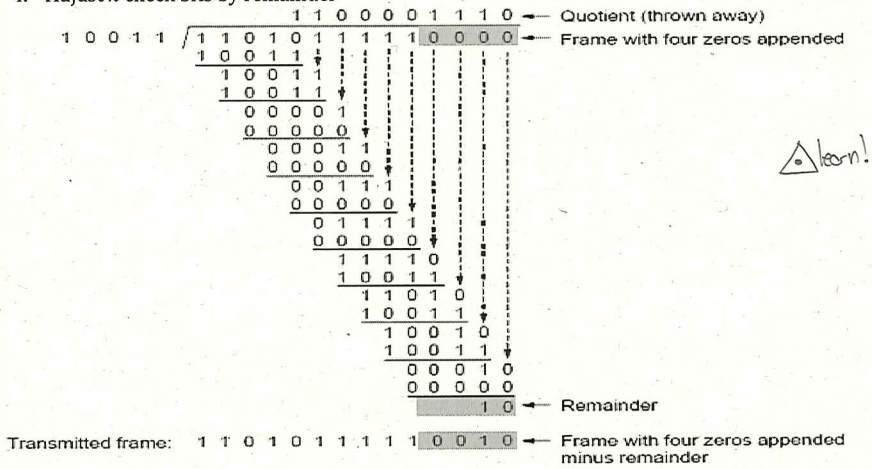
- **Parity bit**: take $D$ data bits, add 1 check bit which is the sum of the $D$ bits (and the modulo 2 or XOR); only little used
- **Checksum**: sum up data in $N$-bit words, stronger protection than parity; used in the Internet
- **Cyclic Redundancy Check (CRC)**: even stronger (unlike checksums, not vulnerable to systematic errors i.e. moving data around); given $n$ data bits, generate $k$ check bits such that the $n + k$ bits are evenly divisible by a generator[2] $C$. It's based on mathematics of finite fields, in which "numbers" represent polynomials meaning we work with binary values and operate using modulo 2 arithmetic; used on links

The internet checksum works as follows:

| SENDING | RECEIVING |
|---|---|
| 1. Arrange data in 16-bit words | 1. Arrange data in 16-bit words |
| 2. Put zero in checksum position and add it | 2. Checksum will be non-zero, add it |
| 3. Add any carryover to get 16 bits | 3. Add any carryover to get 16 bits |
| 4. Negate (complement) to get the sum | 4. Negate (complement) and check if == 0 |

The CRC works as follows:

| SENDING | RECEIVING |
|---|---|
| 1. Extend the $n$ data bits with $k$ zeros | 1. Divide and check for zero remainder |
| 2. Divide by the generator value $C$ | |
| 3. Keep remainder, ignore quotient | |
| 4. Adjust $k$ check bits by remainder | |


1 1 0 0 0 0 1 1 1 0 ← Quotient (thrown away)
1 0 0 1 1 / 1 1 0 1 0 1 1 1 1 1 0 0 0 0 ← Frame with four zeros appended
Remainder
Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder

When performing error correction, keep in mind data might be correct but the check bits aren't.

[2] Recall the definition of a generator: The smallest subgroup of a group containing the element $a \in G$ is the **group generated by** $a$, denoted $\langle a \rangle$, defined as $\langle a \rangle := \{a^n \mid n \in \mathbb{Z}\}$, if the group is finite $\langle a \rangle := \{e, a, a^2, \ldots, a^{\mathrm{ord}(a)-1}\}$. This generated group is called **cyclic** and $g$ is called a **generator** of $G$ ($G = \langle g \rangle$). If $g$ is a generator, so is $g^{-1}$; e.g. $\langle \mathbb{Z}, +, -, 0 \rangle$, $\langle \mathbb{Z}_n, \oplus \rangle$





The **Hamming Code** is a method for constructing a code with a distance of 3. It uses $n = 2^k - k - 1$. It puts check bits in positions p that are powers of 2, starting with position 1 and then the check bit in position p is the parity of positions with a p term in their values. To decode, proceed as follows:

- Recompute check bits (with parity sum including the check bit)
- Arrange as a binary number
- Value (syndrome) tells error position
- Value of zero means no error
- Otherwise, flip bit to correct

An example with $n = 4, k = 3$:

Example: data=0101, 3 check bits
- 7 bit code, check bit positions 1, 2, 4
- Check 1 covers positions 1, 3, 5, 7
- Check 2 covers positions 2, 3, 6, 7
- Check 4 covers positions 4, 5, 6, 7

0 1 0 0 1 0 1 →
1 2 3 4 5 6 7

$p_1 = 0+1+1 = 0$, $p_2 = 0+0+1 = 1$, $p_4 = 1+0+1 = 0$

Example, continued

→ 0 1 0 0 1 0 1
1 2 3 4 5 6 7

$p_1 = 0+0+1+1 = 0$, $p_2 = 1+0+0+1 = 0$, $p_4 = 0+1+0+1 = 0$

Syndrome = 000, no error
Data = 0 1 0 1

Example, continued

→ 0 1 0 0 1 ~~0~~ 1
1 2 3 4 5 6 7

$p_1 = 0+0+1+1 = 0$, $p_2 = 1+0+0+1 = 0$, $p_4 = 0+1+1+1 = 1$

Syndrome = 1 1 0, flip position 6
Data = 0 1 0 1 (correct after flip!)

Other error correcting codes include:

- **Convolutional codes**: take a stream of data and output a mix of the recent input bits; makes each output bit less fragile
- **LDPC**: based on sparse matrices and decoded iteratively using a belief propagation algorithm; *state of the art*

**Detection vs. Correction** depends on the pattern of error and overhead depends, too.

| DETECTION | CORRECTION |
|---|---|
| *In General* | |
| - Needed when errors are expected<br>- Small number of errors are correctable<br>- Or when no time for retransmission<br>- Used in the link layer and above for residual errors (with retransmission) | - More efficient when errors are not expected<br>- And when errors are large when they do occur<br>- Heavily used in physical layer |
| *Assume bit errors are random; messages have 0 or maybe 1 error* | |
| Need ~10 check bits per message | Need ~1 check bit per message plus 1000 bit retransmission 1/10 of the time |
| *Assume errors come in bursts of 100 consecutively garbled bits: only 1 or 2 messages in 1000 have errors* | |

| Need ≫100 check bits per message | Can use 32 check bits per message plus 1000 bit resend 2/1000 of the time |
|---|---|

## Retransmission

If an error is detected, retransmission of that frame can be requested using an **Automatic Repeat Request (ARQ)**. ARQ is often used when errors are common or must be corrected (WiFi, TCP). The rules are: receiver automatically acknowledges correct frames with an ACK; sender automatically resends after a timeout, until an ACK is received. However, there are two non-trivial issues:

- How long to set the timeout? Not too big, not too small; easy on LAN, difficult over the Internet
- How to avoid accepting duplicate frames as new frames? Frames and ACKs must both carry sequence numbers for correctness. It allows only a single frame tbe outstanding from the sender. **Sliding Window** is a generalization of stop-and-wait and allows W frames to be outstanding

## Multiple Access

Multiplexing means sharing resources, classically sharing a link among different users either as:

- **Time Division Multiplexing (TDM)**: users take turns on a fixed schedule, i.e. a user sends at a high rate a fraction of the time
- **Frequency Division Multiplexing (FDM)**: put different users on different frequency bands, i.e. a user sends at a low rate all the time

Both statically divide a resource which is suited for continuous traffic, fixed number of users. However, since network traffic is bursty, TDM/FDM is inefficient. Multiple access schemes multiplex users according to their demands – for gains of statistical multiplexing. This can be (1) randomized (nodes randomize their resource access attempts; good for low load situations) or (2) contention-free (nodes order their resource access attempts; good for high load or guaranteed quality of service situations).

Case study: **ALOHA Network**: Simple idea: node just sends when it has traffic. If there was a collision (no ACK received) then wait a random time and resend. Simple, decentralized protocol that works well under low load but is not efficient under high load: analysis shows at most 18% efficiency. Improvement: divide time into slots and efficiency goes up to 36%.

Another improvement was to listen for activity before sending (which only works with wires), and this is called **Carrier Sense Multiple Access**. This may not be reliable, since another node's sending may be delayed. With **Collision Detection** (CD) the cost of collisions can be reduced by detecting and aborting them (jam). To announce a collision to all parties involved, a minimum frame size of $2D$ seconds is imposed (a node may hear of a collision in $2D$ seconds). When another node is sending, other nodes are waiting and they queue up and then collide. To prevent this from happening, the next sender is chosen with a $1/N$ probability (among $N$ senders).

**Binary Exponential Backoff (BEB)** cleverly estimates the probability by doubling the interval for each successive collision and thereby quickly gets large enough to work which makes it very efficient in practice. 1st collision, wait 0 or 1 frame times, 2nd collision, wait from 0 to 3 times, 3rd collision, wait from 0 to 7 times...

**Classic Ethernet** (IEEE 802.3) was the most popular LAN of the 1980s, 1990s and provided up to 10 Mbps over shared coaxial cable, with baseband signals and multiple access with "1-persistent CSMA/CD with BEB". The Ethernet Frame Format has an addresses to identify the sender and





receiver, uses CRC-32 for error detection; no ACKs or retransmission, and the start of frame identified with physical layer preamble.

Packet from Network layer (IP)

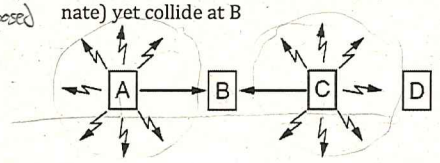
| | Preamble | Destination address | Source address | Type | Data | Pad | Check-sum |
|---|---|---|---|---|---|---|---|
| Bytes | 8 | 6 | 6 | 2 | 0-1500 | 0-46 | 4 |

Multiple access for **Wireless** is more complicated than the wired case because:

1. Nodes may have different areas of coverage – doesn't fit Carrier Sense: wireless signal is broadcast and received nearby, where there is sufficient SNR (signal/noise ratio)

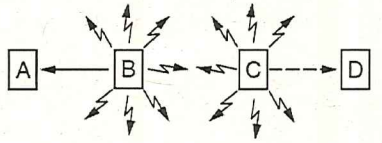
**HIDDEN TERMINALS**

Nodes A and C are hidden terminals when sending to B: can't hear each other (to coordinate) yet collide at B




**EXPOSED TERMINALS**

B and C are exposed terminals when sending to A and D: can hear each other yet don't collide at receivers A and D




2. Nodes can't hear while sending – can't Collision Detect.

A possible solution is **MACA** which uses a short handshake instead of CSMA and works as follows:

1. A sender node transmits a RTS (Request-To-Send, with frame length)
2. The receiver replies with a CTS (Clear-To-Send, with frame length)
3. Sender transmits the frame while nodes hearing the CTS stay silent

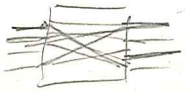
Collisions on the RTS/CTS are still possible, but less likely

**WiFi** i.e. IEEE 802.11 started in the 1990s and clients get connectivity from a (wired) AP (Access Point) and it's also a multi-access problem. It uses 20/40 MHz channels on ISM bands[3] and OFDM modulation (different amplitudes/phases for varying SNRs; rates from 6 to 54 Mbps plus error correction). On the link layer, multiple access uses CSMA/CA; RTS/CTS optional, frames are ACKed and retransmitted with ARQ, errors are detected with a 32-bit CRC. Multiple Access is implemented in such a fashion that the sender avoids collisions by inserting small random gaps, e.g. when both B and C send, C picks a smaller gap, and goes first.

Packet from Network layer (IP)

| | Frame control | Duration | Address 1 (recipient) | Address 2 (transmitter) | Address 3 | Sequence | Data | Check sequence |
|---|---|---|---|---|---|---|---|---|
| Bytes | 2 | 2 | 6 | 6 | 6 | 2 | 0–2312 | 4 |

[3] 802.11b/g/n on 2.4 GHz, 802.11 a/n on 5 GHz



**Contention-Free Multiple Access** is a new approach to multiple access which is based on turns, not randomization. The issues with Random Multiple Access are while CSMA is good under low load (grants immediate access, little overhead (few collisions)) it doesn't perform well under high load (high overhead (expect collisions), access time varies (lucky/unlucky)). That's where **Turn-Taking Multiple Access Protocols** come into play. They define an order in which nodes get a chance to send (or pass, if no traffic at present). The ordering can be done using a Token Ring where nodes are arranged in a ring; token rotates "permission to send" to each node in turn. The advantages are a fixed overhead with no collisions (it's more efficient under load) and a regular chance to send with no unlucky nodes (predictable service, easily extended to guaranteed quality of service) but the main disadvantage is complexity since more things that can go wrong than random access protocols (e.g. what if the token is lost?, higher overhead at low load). It is regularly tried as an improvement offering better service, but random multiple access is hard to beat since it's simple, and usually good enough and scales from few to many nodes.

### Switching

Instead of using multiple access, switches can connect nodes, too, by using multiple links. This is the basis of modern Ethernet. Hosts are wired to Ethernet switches with twisted pair and the switch serves to connect the hosts.

Inside of the switch, all ports are wired together; more convenient and reliable than a single shared wire. It uses frame addresses to connect input port to the right output port, multiple frames may be switched in parallel. A port may be used for both input and output (full-duplex). There is also the need for buffers for multiple inputs to send to one output. Sustained overload will fill buffer and lead to frame loss.

Switches and hubs have replaced the shared cable of classic Ethernet since they're convenient to run wires to one location and more reliable → wire cut is not a single point of failure that is hard to find. Additionally, switches offer scalable performance e.g. 100 Mbps per port instead of 100 Mbps for all nodes of shared cable/hub.

Switch needs to find the right output port for the destination address in the Ethernet frame (**forwarding**). A switch forwards frames with a port/address table as follows:

1. To fill the table, it looks at the source address of input frames
2. To forward, it sends to the port, or else broadcasts to all ports

To deal with a **loop** in the topology (which can either be redundancy in case of failures or a simple mistake), switches collectively find a spanning tree for the topology. The **spanning tree** is a subset of links that is a tree (no loops) and reaches all switches. Switches then forward as normal but only on spanning tree and broadcasts will go up to the root of the tree and down all the branches. The algorithm starts with no information, sends messages, and always searches for the best (and highly robust i.e. adaptive, and configuration-free) solution.

1. Elect a root node of the tree (switch with the lowest address)
2. Grow tree as shortest distances from the root (using lowest address to break distance ties)
3. Turn off ports for forwarding if they are not on the spanning tree

With the following details:

- Each switch initially believes it is the root of the tree
- Each switch sends periodic updates to neighbors with: its address, address of the root, and distance (in hops) to root as a tuple: `(address, root, hops)`
- Switches favors ports with shorter distances to lowest root: uses lowest address as a tie for distances





Switches, however have a few shortcomings (which leads to the introduction of Layer 3):

1. Don't scale to large networks: blow up of routing table, broadcast
2. Don't work across more than one link layer technology: hosts on Ethernet + 3G + 802.11...
3. Don't give much traffic control: want to plan routes / bandwidth

## 3 Network Layer

**Routing** is the process of deciding in which direction to send traffic while **forwarding** is the process of sending a packet on its way.

### Network service models

There are two network service models: (1) datagrams (connectionless), comparable to letters and (2) virtual circuits (connection-oriented), like a phone call. Both models are implemented with store-and-forward packet switching. Routers receive a complete packet, storing it temporarily if necessary before forwarding it onwards. Statistical multiplexing is used to share link bandwidth over time.

When using **datagrams**, packets contain a destination address. Each router uses it to forward each packet, possibly on different paths. To do so, each router has a forwarding table keyed by address which gives the next hop for each destination address (and may change over time). IP (Internet Protocol), which is the network layer of the Internet, uses datagrams.

The **virtual circuit model** consists of three phases:

1. Connection establishment, circuit is set up: path is chosen, circuit information stored in routers
2. Data transfer, circuit is used: packets are forwarded along the path
3. Connection teardown, circuit is deleted: circuit information is removed from routers

Packets only contain a short label to identify the circuit and labels don't have any global meaning, only unique for a link. Each router has a forwarding table keyed by circuit which gives the output line and next label to place on packet. **MPLS** (Multi-Protocol Label Switching) is a virtual-circuit like technology widely used by ISPs where the ISP sets up circuits inside their backbone ahead of time. The ISP adds MPLS label to IP packet at ingress, undoes at egress.

Datagrams and virtual circuits have complementary strengths:

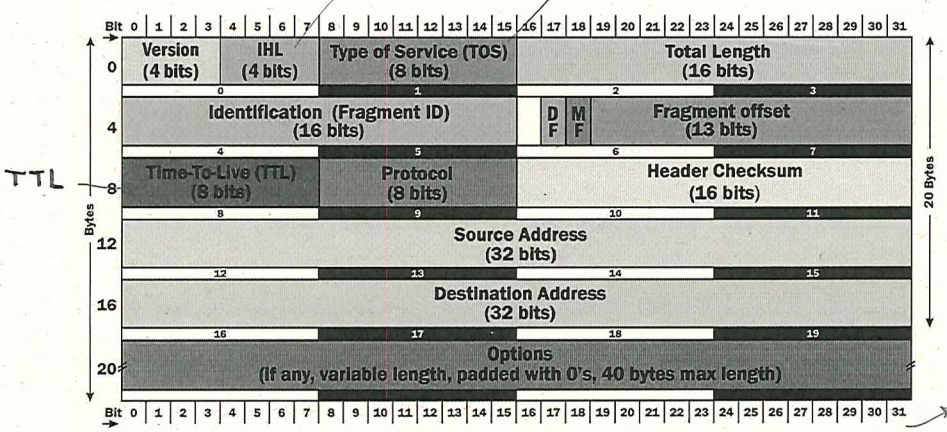
| ISSUE | DATAGRAMS | VIRTUAL CIRCUITS |
|---|---|---|
| SETUP PHASE | Not needed | Required |
| ROUTER STATE | Per destination | Per connection |
| ADDRESSES | Packet carries full address | Packet carries short label |
| ROUTING | Per packet | Per circuit |
| FAILURES | Easier to mask | Difficult to mask |
| QUALITY OF SERVICE | Difficult to add | Easier to add |

IP

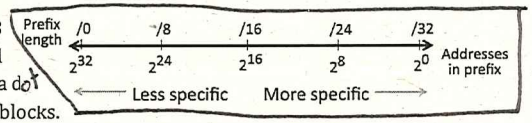
Internetworking is the process of connecting different networks [possible differences: service model (datagrams, VCs), addressing (what kind), QOS (priorities, no priorities), packet sizes, security (whether encrypted)] together. QOS

IP is the lowest common denominator –while some networks may support QOS, internetworking with other networks is difficult. IP achieves this role by asking little of lower-layer networks but only giving a little as a higher layer service.

The **IPv4** packet header:[4]


Bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Version (4 bits)
IHL (4 bits)
Type of Service (TOS) (8 bits)
Total Length (16 bits)
Identification (Fragment ID) (16 bits)
D F
M F
Fragment offset (13 bits)
Time-To-Live (TTL) (8 bits)
Protocol (8 bits)
Header Checksum (16 bits)
Source Address (32 bits)
Destination Address (32 bits)
Options (If any, variable length, padded with 0's, 40 bytes max length)
Bytes
20 Bytes
TTL
differentiated services
then comes the payload

IPv4 uses 32-bit addresses (IPv6: 128 bit) which are written in dotted quad notation, i.e. after every 8 bits, there is a dot. The addresses are allocated in **prefix** blocks.




Addresses in an $L$-bit prefix have the same top $L$ bits. There are $2^{32-L}$ addresses aligned on $2^{32-L}$ boundary. These prefixes are written in "IP address/length $L$" notation where the address is lowest address in the prefix, length is prefix bits. Intuition: the smaller $L$ is, the larger the number of address / the size of the address space.

Most of the IP addresses are **public** (but now exhausted → IPv6), but some address spaces are declared **private**: `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`. These addresses need public IP address(es) and NAT to connect to global Internet.

Allocating Public IP Addresses follows a hierarchical process. The Internet Assigned Numbers Authority (IANA) delegates to regional bodies (RIRs) (e.g. ARIN, RIPE) and they in turn delegate to companies in their region. These companies then assign to their customers/computers. IANA RIR

IP takes care of **forwarding**. IP addresses on one network belong to the same prefix. A node uses a table that lists the next hop for IP prefixes. Prefixes in the table might overlap (this combines

[4] Taken from http://www.v6edu.com/joomla/sixscape/index.php/technical-backgrounders/tcp-ip/ip-the-internet-protocol/ipv4-internet-protocol-version-4/ipv4-packet-header for better legibility; contents are the same as in the lecture slides.



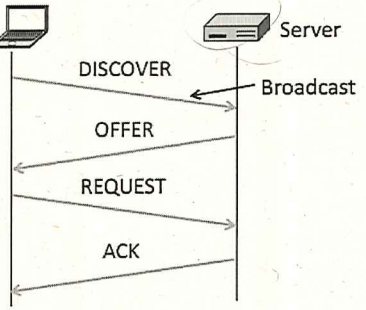


hierarchy with flexibility). Longest matching prefix forwarding rule: for each packet, find the longest prefix that contains the destination address, i.e., the most specific entry and then forward the packet to the next hop router for that prefix. Routers know the way to all destinations and hosts send remote traffic (based on prefix) to the nearest router. 0.0.0.0/0 is a default route that catches all IP addresses (and sends it to the router)[5]. The Longest Matching Prefix rule can provide default behavior, with less specific prefixes (to send traffic going outside an organization to a border router) and can also provide special case behavior, with more specific prefixes (for performance, economics, security). While finding the longest match more complex than table lookup, it's of no concern in practice. Other aspects when forwarding (apart from the address):

- Decrement TTL value: protects against loops
- Checks header checksum: to add reliability
- Fragment large packets: split to fit it on next link
- Send congestion signals: warns hosts of congestion
- Generates error messages: to help manage network
- Handle various options

DHCP

**DHCP (Dynamic Host Configuration Protocol)** is the process of getting an IP address and establishing connection to the router for the first time. It leases IP address to nodes and also provides the network prefix, the address of local router, DNS server, time server etc. DHCP is a client-server application on UDP (use datagram protocol) ports 67 and 68. To send a message to DHCP server before it is configured, the node sends broadcast messages that are delivered to all nodes on the network. The broadcast address in IP is 255.255.255.255. DHCP also supports renewing leases (which only consist of a REQUEST and an ACK) and replicated servers for reliability.




ARP

**ARP** maps IP to link addresses. A node uses ARP to map a local IP address to its Link layer addresses. ARP sits right on top of link layer and doesn't use any servers, just asks node with target IP to identify itself and uses broadcast to reach all nodes. IP → MAC address

MTU

To connect networks with different **maximum packet sizes** / MTU (Maximum Transmission Unit), some sort of mechanism is needed. Larger MTUs are generally preferred (for efficiency). This is difficult because node does not know complete network path. **Fragmentation** splits up large packets in the network if they are too big to send. This method is a bit dated. Nowadays, **discovery** is used which the largest packet that fits on the network path and uses it.

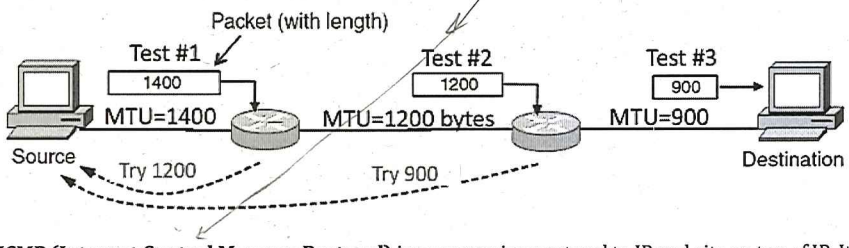
When using **fragmentation**, the routers fragment packets that are too large to forward and the receiving host reassembles to reduce load on routers. Header fields (identification, fragment offset, MF/DF control bits) are used to handle packet size differences. While it works, it imposes more work on routers, magnifies severity of packet loss, and also has security vulnerabilities.

MF DF

more fragments / don't fragment

[5] A very simple host forwarding table contains two entries (prefix, next hop): (own network prefix, that IP) and (0.0.0.0/0, to router).

**Path MTU Discovery** works by discovering the MTU that will fit to avoid fragmentation. Host tests path with large packet and routers provide feedback if too large; they tell host what size would have fit. While this seems a bit "too much", it's usually quick. Path MTU depends on the path, so it can change over time. It's implemented with ICMP and sets DF (Don't Fragment) bit in IP header to get feedback messages.




**ICMP (Internet Control Message Protocol)** is a companion protocol to IP and sits on top of IP. It provides error report and testing. When a router encounters an error while forwarding it sends an ICMP error report back to the IP source address and discards the problematic packet (the host needs to rectify). Each ICMP message has a(type, code, and checksum)and it often carries the start of the offending packet as payload. Each message is carried in an IP packet. Examples include:

- Destination unreachable (network or host)
- Destination unreachable (fragment)
- Time exceeded (transit)
- Echo request or reply

The **traceroute** makes use of the TTL field in the IP header. It is decremented every router hop, with ICMP error if it hits zero to protect against forwarding loops. Traceroute repurposes TTL and ICMP functionality by sending probe packets increasing TTL starting from 1 and using ICMP errors to identify routers on the path.

IPv6

IPv6 is the solution to the lack of available IPv4 addresses. It features larger addresses (most of the header is 128 bits) and uses a new notation: 8 groups of 4 hex digits (16 bits), omit leading zeros, groups of zeros (by substituting it with "%"). It also features lots of smaller changes such as streamlined header processing, flow label to group of packets, and better fit with "advanced" features (mobility, multicasting, security). Since it's fundamentally incompatible with IPv4, there are a number of approaches for the transition: dual stack, translator, tunnels (native IPv6 islands connected via IPv4; the tunnel carries IPv6 packets across IPv4 network).

NAT

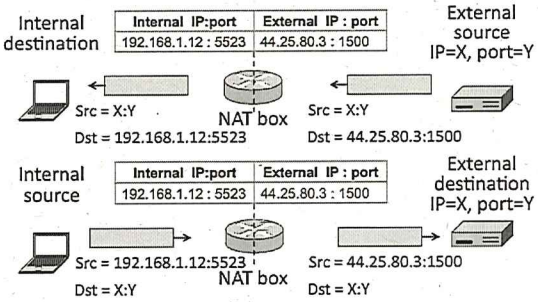
**NAT (Network Address Translation)** is widely used at the edges of the network. Middleboxes sit "inside the network" but perform "more than IP" processing on packets to add new functionality (NAT box, firewall / intrusion detection system).

| ADVANTAGES | DISADVANTAGES |
|---|---|
| - A possible rapid deployment path when there is no other option | - Breaking layering interferes with connectivity; strange side effects |
| - Control over many hosts | - Poor vantage point for many tasks |





NAT box connects an internal network to an external network thus connecting many internal hosts ~~are connected~~ using few external addresses (motivated by IP address scarcity). Commonly home computers use "private" IP addresses and the AT (in AP/firewall) connects home to ISP using a single external IP address. This works by keeping an internal/external table, typically uses IP address + TCP port. Ports are needed to make to mapping 1:1 since there are fewer external IPs.




| DOWNSIDES | UPSIDES |
|---|---|
| - Connectivity has been broken: can only send incoming packets after an outgoing connection is set up; difficult to run servers or peer-to-peer apps (Skype) at home | - Relieves much IP address pressure: many home hosts behind NATs<br>- Easy to deploy: rapidly, and by you alone |
| - Doesn't work so well when there are no connections (UDP apps) | - Useful functionality: firewall, helps with privacy |
| - Breaks apps that unwisely expose their IP addresses (FTP) | - Kinks will get worked out eventually: "NAT Traversal" for incoming traffic |

Routing Algorithms

While the spanning tree provides basic connectivity, routing uses all links to find best paths.

| GOALS | RULES | "BEST" PATHS |
|---|---|---|
| - Correctness: finds paths that work | - Decentralized, distributed setting | - Latency, avoid circuitous paths |
| - Efficient paths: uses network bandwidth well | - All nodes are alike; no controller | - Bandwidth, avoid small pipes |
| - Fair paths: doesn't starve any nodes | - Nodes only know what they learn by exchanging messages with neighbors | - Money, avoid expensive links |
| - Fast convergence: recovers quickly after changes | | - Hops, to reduce switching |
| - Scalability: works well as network grows large | - Nodes operate concurrently<br>- May be node/link/message failures | But only consider topology i.e. ignore workload, e.g., hotspots |

Shortest path routing

Shortest path routing associates links with a cost function to capture the factors of "best-ness".

1. Assign each link a cost (distance)
2. Define best path between each pair of nodes as the path that has the lowest total cost (or is shortest)
3. Pick randomly to any break ties

The **sink tree** for a destination is the union of all shortest paths towards the destination Thus, only the destination is needed to follow shortest paths and each node only need to send to the

next hop. There is a forwarding table at a node which lists the next hop for each destination. The routing table may know more.

To compute the shortest Path, Dijkstra's algorithm can be used:

| ALGORITHM | COMMENTS |
|---|---|
| - Mark all nodes tentative, set distances from source to 0 for source, and ∞ for all other nodes<br>- While tentative nodes remain:<br>  ▪ Extract N, a node with lowest distance<br>  ▪ Add link to N to the shortest path tree<br>  ▪ Relax the distances of neighbors of N by lowering any better distance estimates | - Finds shortest paths in order of increasing distance from source: leverages optimality property<br>- Runtime depends on efficiency of extracting min-cost node: superlinear in network size (grows fast)<br>- Gives complete source/sink tree which is more than needed for forwarding but requires complete topology |

### Distance Vector routing

**Distance Vector (DV)** routing is a simple, early routing approach and also used in the RIP (Routing Information Protocol).It uses a distributed version of Bellman-Ford which works, but very slow convergence after some failures. Each node computes its forwarding table in a distributed setting:

- Nodes know only the cost to their neighbors; not the topology
- Nodes can talk only to their neighbors using messages
- All nodes run the same algorithm concurrently
- Nodes and links may fail, messages may be lost

Each node maintains a vector of distances (and next hops) to all destinations:

1. Initialize vector with 0 cost to self, ∞ to other destinations
2. Periodically send vector to neighbors
3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
- Use the best neighbor for forwarding

When adding routes, news travels one hop per exchange. When a node fails, no more exchanges, other nodes forget. The problem are partitions (unreachable nodes in divided network) where a "count to infinity" scenario happens.

**RIP (Routing Information Protocol)** uses the DV protocol with hop count as metric and defines infinity to be 16 hops which limits network size. It includes split horizon, poison reverse[6]. Routers send vectors every 30 secs and uses a timeout of 180 secs to detect failures.

Due to various issues (e.g. split horizon, poison reverse), DV is now often replaced by link state in LAN settings (except when very resource-limited).

### Flooding

**Flooding** broadcasts a message to all nodes in the network. Rule used at each node:

- Sends an incoming message on to all other neighbors

[6] Don't send route back to where you learned it from





- Remember the message so that it is only sent once over each link (called duplicate suppression)

Inefficient because one node may receive multiple copies of message.

Remember message (to stop flood) using source and sequence number. This is used for duplicate suppression, so same message is only sent once to neighbors. This means, subsequent message (with higher sequence number) will again be flooded. To make flooding reliable, use ARQ i.e. the receiver acknowledges, and sender resends if needed.

### Link-state routing

**Link-state routing** trades more computation than distance vector for better dynamics and is widely used in practice. Each node computes their forwarding table in the same distributed setting as distance vector and proceeds in two phases:

1. Nodes flood topology in the form of link state packets (LSPs): each node learns full topology
2. Each node computes its own forwarding table by running Dijkstra (or equivalent)

On change, flood updated LSPs, and re-compute routes. When a link fails, both nodes notice, send updated LSPs and the link is removed from topology. When a node fails, all neighbors notice a link has failed and even though the failed node can't update its own LSP, all links to node are removed. When a link or node is added, LSP of new node is added to the topology and old LSPs are updated with the new link. Complications arise when:

- Seq. number reaches max, or is corrupted
- Node crashes and loses seq. number
- Network partitions then heals

This is dealt with including an age on LSPs and forget old information that is not refreshed.

While DV scales really well, Link-State converges really fast.

### Equal-cost multi-path

**Equal-Cost Multi-Path Routing** allows for multiple shortest paths. This allows for multiple routing paths from node to destination be used at once which means the topology has them for redundancy and using them can improve performance and reliability. One form of multipath routing extends shortest path model by keeping them set if there are ties. With ECMP, source/sink "tree" is a directed acyclic graph (DAG).

Forwarding with ECMP works by trying to send packets from a given source/destination pair on the same path: A source/destination pair is called a flow. Map flow identifier to single next hop. This results in no jitter within flow, but less balanced than randomly picking a next hop for each packet based on.

### Hierarchical Routing

This is about routing to regions, not individual nodes. This introduces a larger routing unit by first routing to the region and then to the IP prefix within that region. It hides details within a region from outside of the region. Outside a region, nodes have one route to all hosts within the region.

This gives savings in table size, messages and computation. However, each node may have a different route to an outside region– Routing decisions are still made by individual nodes and there is no single decision made by a region.

Two use cases for adjusting the size of IP prefixes; both reduce routing table size:

- **Subnets:** internally split one less specific prefix into multiple more specific prefixes
- **Aggregation:** externally join multiple more specific prefixes into one large prefix

**Routing with Multiple Parties & Inter-domain routing (BGP)** → border gateway protocol

Networks (ISPs, CDNs, etc.) group hosts as IP prefixes and these networks are richly interconnected, often using IXPs. There are two issues when it comes to routing:

1. Scaling to very large networks
2. Incorporating policy decisions

There are two routing policies: PEER and TRANSIT:

| TRANSIT | PEER |
|---|---|
| One party (customer) gets TRANSIT service from another party (ISP) | Both party (ISPs in example) get PEER service from each other |
| - ISP accepts traffic from customer to deliver to the rest of Internet<br>- ISP accepts traffic from the rest of the Internet to delivery to customer<br>- Customer pays ISP for the privilege | - Each ISP accepts traffic from the other ISP only for their customers<br>- ISPs do not carry traffic to the rest of the Internet for each other<br>- ISPs don't pay each other |

**BGP** computes Internet-wide routes, using path vectors (similar to a DV). Different parties like ISPs are called AS (Autonomous Systems). Border routers of ASes announce BGP routes to each other. The route announcements contain an IP prefix, path vector, next hop. The path vector is a list of ASes on the way to the prefix; list is to find loops. Route announcements move in the opposite direction to traffic. Policy is implemented in two ways:

1. Border routers of ISP announce paths only to other parties who may use those paths: filter out paths others can't use
2. Border routers of ISP select the best path of the ones they hear in any, non-shortest way

## 4 Transport Layer

Service Models

| TCP (STREAMS) ✓ | UDP (DATAGRAMS) ✗ |
|---|---|
| Connections | Datagrams |
| Bytes are delivered once, reliably, and in order | Messages may be lost, reordered, duplicated |
| Arbitrary length content | Limited message size |
| Flow control matches sender to receiver | Can send regardless of receiver state |
| Congestion control matches sender to network | Can send regardless of network state |

The **Socket API** is a simple abstraction to use the network and part of all major OSes. It supports both streams and datagrams. IT allows apps to attach to the local network at different **ports.**



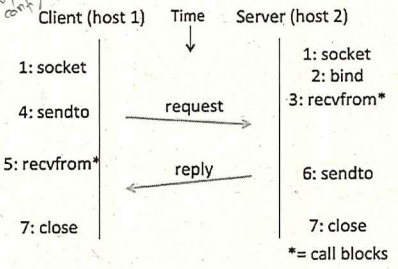


| PRIMITIVE | MEANING | STREAM (S) OR DATAGRAM (D) ONLY |
|---|---|---|
| SOCKET | Create a new communication endpoint | |
| BIND | Associate a local address (port) with a socket | |
| LISTEN | Announce willingness to accept connections | S |
| ACCEPT | Passively establish an incoming connection | S |
| CONNECT | Actively attempt to establish a connection | S |
| SEND(TO) | Send some data over the socket | D |
| RECEIVE(FROM) | Receive some data over the socket | D |
| CLOSE | Release the socket | |

**Ports** are 16-bit integers, comparable to local mailboxes. An aplication process is identified by the tuple (IP address, protocol, and port) While servers often bind to "well-known ports" (<1024, require administrative privileges), clients are often assigned "ephemeral" ports (chosen by OS, used temporarily).

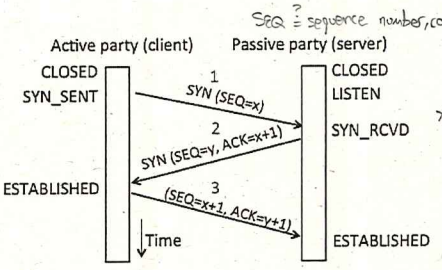
User Datagram Protocol (UDP)

UDP is used by apps that don't want reliability or byte streams, such as voice-over-IP (unreliable), DNS, RPC (message-oriented), or DHCP (bootstrapping). The UDP header uses ports to identify sending and receiving application processes. The length of the datagram length is up to 64K and there's a checksum (16 bits) for reliability. The UDP Pseudoheader uses an optional checksum which covers the UDP segment and IP fields. A value of zero means "no checksum".



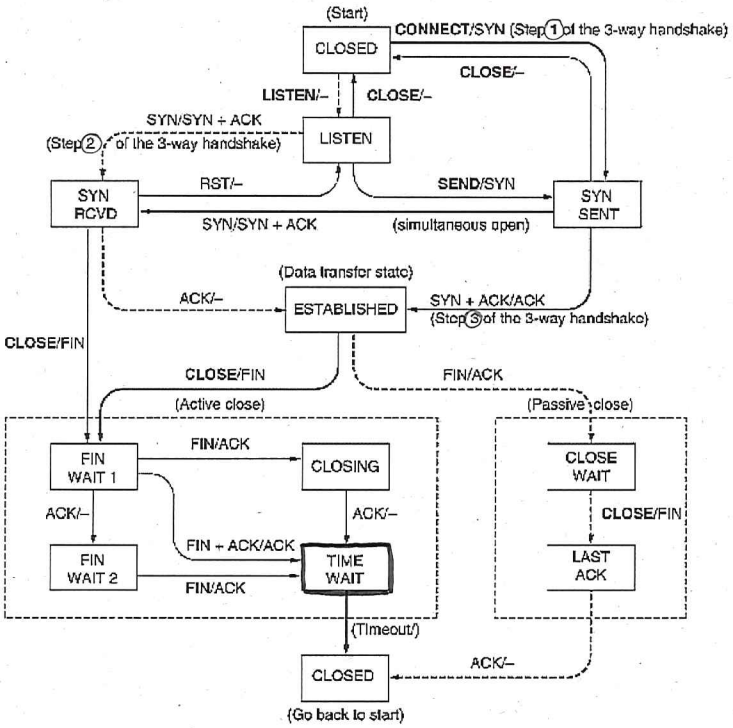

Connections (TCP)

Both sender and receiver must be ready before the transfer of data is started. Both parties need to agree on a set of parameters e.g., the Maximum Segment Size (MSS). This is an example of signaling, it sets up state at the endpoints. TCP uses a **three-way-handshake** which opens connection for data in both directions. Each side probes the other with a fresh Initial Sequence Number (ISN). This is robust even against delayed duplicates.

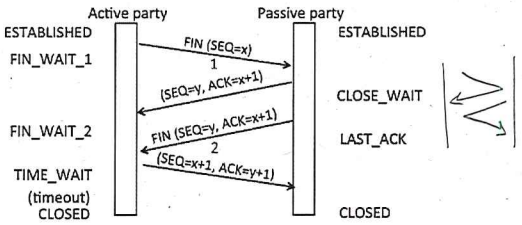



The TCP Connection State Machine looks as follows where A/B means event A triggers the transition, with action B.[7]

[7] Same as in the lecture slides; taken from http://jose-manuel.me/wp-content/uploads/2011/09/connection-fsm.gif




A **connection release** is an orderly release by both parties when done. First, all pending data is delivered and then it "hangs up" i.e. it cleans up state in sender and receiver. The key problem is to provide reliability while releasing which TCP handles with a "symmetric" close in which both sides shutdown independently.




In the **TIME_WAIT** state, TCP waits a long time (two times the maximum segment lifetime of 60 seconds) after sending all segments and before completing the close. This is done in case the ACK was lost, in which case FIN will be resent for an orderly close. Otherwise, this could interfere with a subsequent connection.

### Sliding Window (TCP)

Stop-and-Wait only allows a single message to be outstanding from the sender which is fine for LAN but not efficient for network paths with BD[8] $\gg 1$ packet. An example: $R = 1$ Mbps, $D =$ 50 ms, $RTT = 2D = 100$ms. Packets per second: 10 which is a 10% utilization. If $R = 10$Mbps, there are still 10 packets per second but the utilization is around 1%.

[8] Recall: bandwidth-delay





**Sliding Window** is a generalization of stop-and-wait and allows $W$ packets to be outstanding and can send $W$ packets per $RTT$ ($= 2D$). Pipelining improves performance and needs $W = 2BD$ to fill the network path.

Example: determine the $W$ to fill the network path:

- $R = 1$Mbps, $D = 50$ms → $2BD = 10^6$b/s $\cdot\ 100 \cdot 10^{-3}$s $= 100$ kbit ; W = 2BD = 10 packets of 1200 bytes
- $R = 10$Mbps, $D = 50$ms → $2BD = 10^7$b/s $\cdot\ 100 \cdot 10^{-3}$s $= 1000$ kbit ; W = 2BD = 100 packets of 1200 bytes

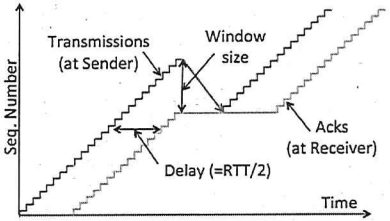
The protocol has many variations, depending on how buffers, acknowledgements, and retransmissions are handled. Go-Back-N is the simplest version and can be inefficient while **Selective** Repeat is more complex but has better performance.

Using **Go-Back-N** the receiver keeps only a single packet buffer for the next segment (state variable, LAS = LAST ACK SENT) and on receive: if seq. number is $LAS + 1$, accept and pass it to app, update $LAS$, send $ACK$; otherwise discard (as out of order).

Using **Selective Repeat** the receiver passes data to app in order, and buffers out-of-order segments to reduce retransmissions. ACK conveys highest in-order segment, plus hints about out-of-order segments. TCP uses a selective repeat design. Buffers $W$ segments, keeps state variable LAS. On receive: there are two buffer segments $[LAS + 1, LAS + W]$: pass up to app in-order segments from $LAS + 1$, and update $LAS$ and send ACK for LAS regardless.

For **retransmission** Go-Back-N sender uses a single timer to detect losses and on timeout, resends buffered packets starting at $LAR + 1$. Selective Repeat sender uses a timer per unacked segment to detect losses and on timeout for a segment, resend it in the hope of resending fewer segments.

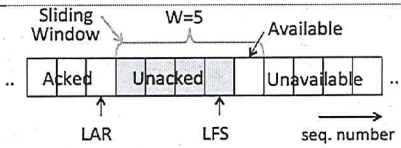
**Sequence numbers** play an important role. For Selective Repeat, $W$ numbers for packets are need, plus $W$ for ACKs of earlier packets i.e. $2W$ seq. numbers. Fewer numbers are needed for Go-Back-N ($W + 1$). Typically, sequence numbers are implemented with an $N$-bit counter that wraps around at $2^N - 1$, e.g., N=8: ..., 253, 254, 255, 0, 1, 2, 3, ...



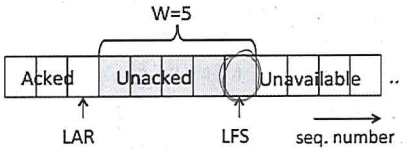

**SENDER**

Sender buffers up to $W$ segments until they are acknowledged. Sends while $LFS - LAR \leq W$

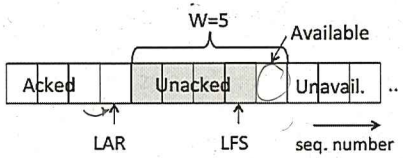
LFS=last frame sent, LAR=last ACK rec'd




Transport accepts another segment of data from the Application... Transport sends it (as $LFS - LAR \rightarrow 5$)

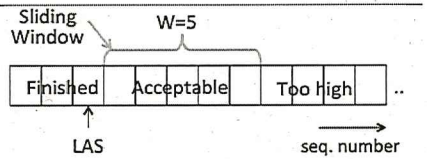


Next higher ACK arrives from peer... Window advances, buffer is freed; $LFS - LAR \rightarrow 4$ (can send one more).



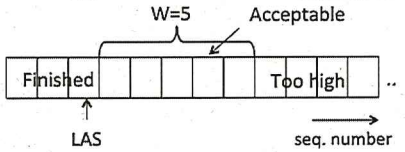

**RECEIVER**

Consider receiver with $W$ buffers, app pulls in-order data from buffer with recv() call.

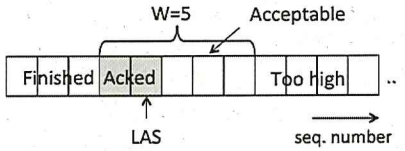



recv()

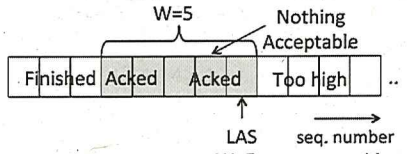
Suppose the next two segments arrive but app does not call recv().



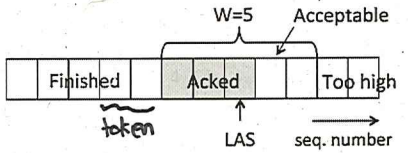

LAS rises, but the window can't slide




If further segments arrive (even in order) the buffer can be filled but it has to drop segments until app recv()s.

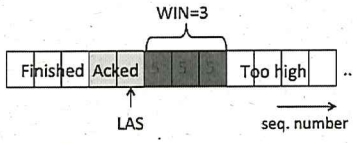



App recv() takes two segments and the window slides




### Flow Control

Sliding window uses pipelining to keep the network busy but also has to do some **flow control** to slow the over-enthusiastic sender. Loss at the receiver is avoided by telling the sender the available buffer space. The sender uses the lower of the sliding window and flow control window (WIN) as the effective window size.




### Retransmission Timers

With sliding window, the strategy for detecting loss is the timeout. A timer is set when a segment is sent and cancelled when ACK is received. If the timer fires, retransmit data as lost. The timeout should be neither too long since that wastes network capacity nor too short because leads to spurious resends. On LAN (Link) the timeout is short and fixed, but on the Internet (transport)





there's a wide range. **Adaptive Timeout** keeps smoothed estimates of the RTT and variance in RTT. It's simple to compute and does a good job of tracking actual RTT.

### Transmission Control Protocol (TCP)

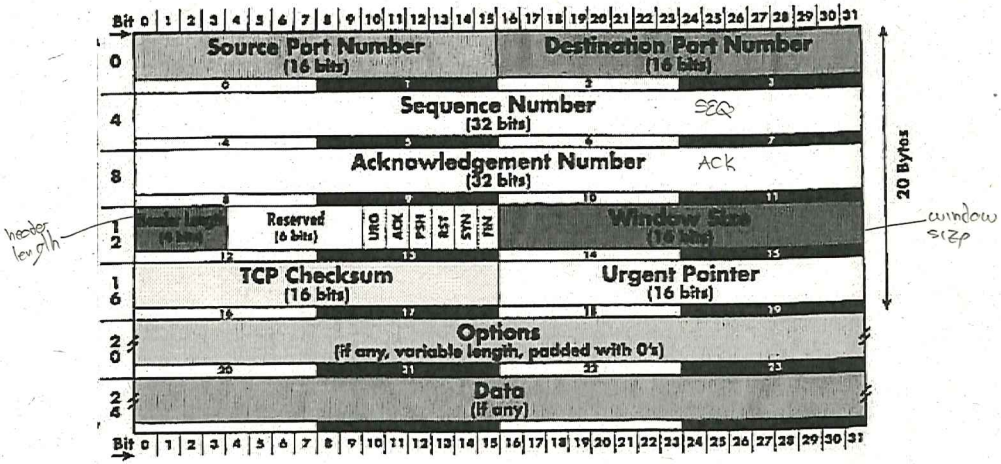
TCP provides a **reliable bytestream** meaning while message boundaries not preserved from send() to recv(), the messages are reliable and ordered (receive bytes in same order as sent). It also implements a bidirectional data transfer thus control information (e.g. ACK) piggybacks on data segments in reverse direction.

The **TCP Header**:[9]

- Ports identify apps (socket API); 16-bit identifiers
- SEQ/ACK used for sliding window; Selective Repeat, with byte positions
- SYN/FIN/RST flags for connections; flag indicates segment is a SYN etc.
- Window size for flow control; relative to ACK, and in bytes

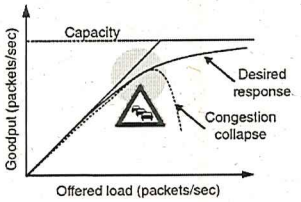
TCP **Sliding Window** on the *receiver* side: cumulative ACK tells next expected byte sequence number ($LAS + 1$) and optionally, selective ACKs (SACK) give hints for receiver buffer state, List up to 3 ranges of received bytes. On the *sender* side: uses adaptive retransmission timeout to resend data from $LAS + 1$ Uses heuristics to infer loss quickly and resend to avoid timeouts. Three duplicate ACKs are treated as loss.

SACK


Bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Source Port Number (16 bits)
Destination Port Number (16 bits)
Sequence Number (32 bits)
Acknowledgement Number (32 bits)
Reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window Size (16 bits)
TCP Checksum (16 bits)
Urgent Pointer (16 bits)
Options (if any, variable length, padded with 0's)
Data (if any)
20 Bytes
header length
window size

### Nature of congestion

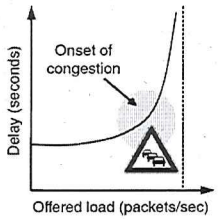
Routers/switches have internal buffering for contention. This queues are typically FIFO and discarded when full. Queues help by absorbing bursts when input > output rate. But if input > output rate persistently, the queue will overflow → congestion. Congestion is a function of the traffic patterns – can




[9] Taken from https://baltazaar.files.wordpress.com/2013/04/tcp-header.jpg?w=614

occur even if every link have the same capacity. As offered load rises, congestion occurs as queues begin to fill:

- Delay and loss rise sharply with more load
- Throughput falls below load (due to loss)
- Goodput may fall below throughput (due to spurious retransmissions)




### Bandwidth Allocation / Fair allocations

Important task for the network is to allocate its capacity to senders – a good allocation is efficient and fair. *Efficient* means most capacity is used but there is no congestion. *Fair* means every sender gets a reasonable share the network. In an effective solution, Transport and Network layers must work together: Network layer witnesses congestion – only it can provide direct feedback; Transport layer causes congestion – only it can reduce offered load.

| CHALLENGE | SOLUTION |
|---|---|
| - The number of senders and their offered load is constantly changing | - Senders adapt concurrently based on their own view of the network |
| - Senders may lack capacity in different parts of the network | - Design this adaption so the network usage as a whole is efficient and fair |
| - Network is distributed; no single party has an overall picture of its state. | - Adaption is continuous since offered loads continue to change over time |

And in practice, efficiency is more important than fairness. It isn't productive to seek exact fairness, it's more important to avoid starvation – "equal per flow" is good enough. The **bottleneck** for a flow of traffic is the link that limits its bandwidth. This is where congestion occurs for the flow.

**Max-Min Fairness** intuitively, flows bottlenecked on a link get an equal share of that link. Thus increasing the rate of one flow will decrease the rate of a smaller flow. This "maximizes the minimum" flow. Allocation changes as flows start and stop.

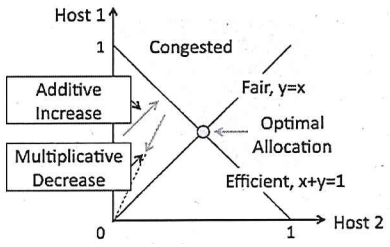
### Bandwidth Allocation Models

- Open loop versus closed loop
  - Open: reserve bandwidth before use
  - Closed: use feedback to adjust rates
- Host versus Network support: who sets/enforces allocations?
- Window versus Rate based: how is allocation expressed?

TCP is closed loop, host-driven, and window-based.

#### Additive Increase Multiplicative Decrease (AIMD) control law

AIMD is a control law hosts can use to reach a good allocation. Hosts *additively increase* rate while network is not congested and hosts *multiplicatively decrease* rate when congestion occurs. This produces a "sawtooth" pattern over time for rate of each host.








AMID converges to an allocation that is efficient and fair when hosts run it and requires only binary feedback from the network.

### Feedback signals

| SIGNAL | EXAMPLE PROTOCOL | PRO | CON |
|---|---|---|---|
| PACKET LOSS | TCP NewReno<br>Cubic TCP (Linux) | Hard to get wrong | Hear about congestion late |
| PACKET DELAY | Compound TCP (Windows) | Hear about congestion early | Need to infer congestion |
| ROUTER INDICATION | TCPs with Explicit Congestion Notification | Hear about congestion early | Require router support |

### TCP Congestion Control history

Early TCP used a fixed size sliding window (e.g., 8 packets) but as the ARPANET grew, links stayed busy but transfer rates fell by orders of magnitude. To avoid congestion collapse without changing routers (or even receivers), the idea was to fix timeouts and introduce a **congestion window** (cwnd) over the sliding window to limit queues/loss. TCP Tahoe/Reno implements AIMD by adapting cwnd using packet loss as the network feedback signal

### ACK clocking

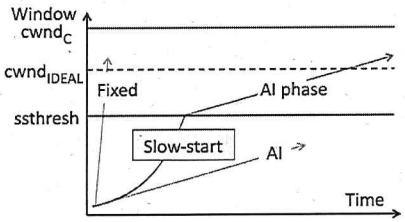
Each in-order ACK advances the sliding window and lets a new segment enter the network thus ACKs "clock" data segments. This helps dealing with bursts: segments are buffered and spread out on slow link; ACKs maintain the spread back to the original sender; sender clocks new segments with the spread – now sending at the bottleneck link without queuing. This helps the network run with low levels of loss and delay. The network has smoothed out the burst of data segments. ACK clock transfers this smooth timing back to the sender and subsequent data segments are not sent in bursts so they do not queue up in the network.

TCP uses ACK clocking and a sliding window because of the value of ACK clocking. Sliding window controls how many segments are inside the network (which is called the congestion window, or cwnd; rate is roughly *cwnd/RTT*). TCP only sends small bursts of segments to let the network keep the traffic smooth.

### TCP Slow-start

"Slow start" is a component of the AI portion of AIMD.

Timeouts are sufficiently long that the ACK clock will have run down. The slow-start ramps up the ACK clock.




| PROBLEMS | SOLUTION |
|---|---|
| - The cwnd$_{IDEAL}$ should be achieved quickly, but it varies greatly | - Start by doubling cwnd every RTT: Exponential growth (1, 2, 4, 8, 16, ...) quickly reaches large values |
| - Fixed sliding window doesn't adapt and is rough on the network | - Eventually packet loss will occur when the network is congested: loss timeout gives a hint when cwnd is too large and next time, switch to AI beforehand: slowly adapt cwnd near right value |
| - AI with small bursts adapts cwnd gently to | - Combined behavior, after first time: most time spend near right value |

the network, but might take a long time to become efficient
- Expect loss for $cwnd_c \approx 2BD + queue$
- Use $ssthresh = cwnd_c/2$ to switch to AI after observing loss
- Slow-Start phase: Increment cwnd by 1 packet for each ACK
- Additive Increase phase: Increment cwnd by 1 packet every cwnd ACKs (or 1 RTT)

### TCP Fast Retransmit/Recovery

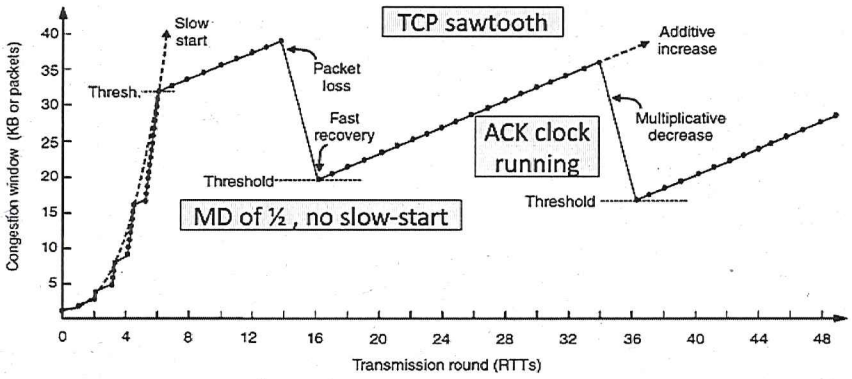
"Fast retransmit" and "fast recovery" are the MD portion of AIMD. Inferring Loss from ACKs is possible since TCP uses a cumulative ACK which carries highest in-order seq. number which normally is a steady advance. Duplicate ACKs give hints about what data hasn't arrived i.e. some new data did arrive, but it was not next segment and thus the next segment may be lost.

**Fast Retransmit** treats three duplicate ACKs as a loss and then retransmits next expected segment. Some repetition allows for reordering, but it still detects a loss quickly. It can repair single segment loss quickly, typically before a timeout

Also non-losses can be inferred from ACKs. Duplicate ACKs also give hints about what data has arrived. Each new duplicate ACK means that some new segment has arrived, to be precise, it will be the segments after the loss. Thus advancing the sliding window will not increase the number of segments stored in the network.

**Fast Recovery** first uses a fast retransmit and MD cwnd. Then it pretends further duplicate ACKs are the expected ACKs which allows new segments be sent for ACKs and reconciles views when the ACK jumps. With fast retransmit, it repairs a single segment loss quickly and keeps the ACK clock running. This allows AIMD to be realized since there are no timeouts or slow-start after a loss, it just continues with a smaller cwnd.

### TCP Reno




### Congestion Avoidance (ECN)

Classic TCP drives the network into congestion and then recovers which means it needs to see loss to slow down.

Using **ECN**, the router detects the onset of congestion via its queue and when congested, it marks affected packets (in the IP header). When a marked packets arrive at receiver, it is treated as loss and the TCP receiver reliably informs TCP sender of the congestion.





| ADVANTAGES | DISADVANTAGES |
|---|---|
| - Routers deliver clear signal to hosts<br>- Congestion is detected early, no loss<br>- No extra packets need to be sent | - Routers and hosts must be upgraded |

## 5 Session Layer

**NOT PART OF THE LECTURE**

## 6 Presentation Layer

**NOT PART OF THE LECTURE**

## 7 Application Layer

### DNS (Domain Name System)

The DNS converts IP addresses into human-readable host names (and some more).

- **Names** are higher-level identifiers for resources
- **Addresses** are lower-level locators for —"—
- **Resolution** (or lookup) is mapping a name to an address

Before DNS, there was a file called HOSTS.TXT which regularly retrieved for all hosts from a central machine at the NIC (Network Information Center)- Names were initially flat and later on became hierarchical.

DNS strives to be easy to manage and be efficient (performance, resources) and does so by using a distributed directory based on a hierarchical namespace and an automated protocol to tie pieces together.

The hierarchy starts from the "." and then goes down to TLDs (top-level domains), "domains", subdomains etc. The TLDs are assigned by the ICANN and country-specific and generic TLDs exist. A **zone** is a contiguous portion of the namespace and are the basis for distribution. Each zone has a nameserver to contact for information about it. A zone is comprised of DNS resource records that give information for its domain names.

| TYPE | MEANING |
|---|---|
| SOA | Start of authority, has key zone parameters |
| A | IPv4 address of a host |
| AAAA ("QUAD A") | IPv6 address of a host |
| CNAME | Canonical name for an alias |
| MX | Mail exchanger for the domain |
| NS | Nameserver of domain or delegated subdomain |

DNS protocol lets a host **resolve** any host name (domain) to an IP address. If unknown, it can start with the root nameserver and work down zones.

| RECURSIVE QUERY | ITERATIVE QUERY |
|---|---|
| - Nameserver completes resolution and returns the final answer<br>- Lets server offload client burden (simple resolver) for manageability<br>- Lets server cache over a pool of clients for better performance | - Nameserver returns the answer or who to contact next for the answer<br>- Lets server "file and forget"<br>- Easy to build high load servers |

Resolution latency should be low since it adds delay to web browsing. Thus, query/responses are cached to answer future queries immediately, including partial (iterative) answers. Responses carry a TTL for caching.[10]

The root (dot) is served by 13 server names and are highly available and reliable. The >250 distributed server instances use IP anycast[11].

The **DNS Protocol** is built on UDP messages and uses port 53. ARQ is used for reliability. The server is stateless. Messages linked by a 16-bit ID field. Service reliability is provided via replicas. Security is a major issue and DNSSEC is only partially deployed.

### HTTP (HyperText Transfer Protocol)

HTTP is a request/response protocol for fetching web resources and runs on TCP, typically port 80 (HTTPS is on 443). Originally a simple protocol, with many options added over time. Commands are text-based and added to the headers. It uses parallel and persistent connections.

The steps involved when fetching a web page over HTTP are:

- Resolve the server to IP address (DNS)
- Set up TCP connection to the server
- Send HTTP request for the page
- (Await HTTP response for the page)
- Execute / fetch embedded resources / render
- Clean up any idle TCP connections

| METHOD | DESCRIPTION |
|---|---|
| GET | Read a Web page |
| HEAD | Read a Web page's header |
| POST | Append to a Web page |
| PUT | Store a Web page |
| DELETE | Remove the Web page |
| TRACE | Echo the incoming request |
| CONNECT | Connect through a proxy |
| OPTIONS | Query options for a page |

| CODE | MEANING | EXAMPLES |
|---|---|---|
| 1XX | Information | 100 = server agrees to handle client's request |
| 2XX | Success | 200 = request succeeded; 204 = no content present |
| 3XX | Redirection | 301 = page moved; 304 = cached page still valid |
| 4XX | Client error | 403 = forbidden page; 404 = page not found |
| 5XX | Server error | 500 = internal server error; 503 = try again later |

**PLT (Page Load Time)** is the key measure of web performance: from click until user sees page. The PLT depends on many factors: structure of page/content, HTTP (and TCP) protocol, network RTT and bandwidth etc. Ways to Decrease PLT are:

[10] And that's why, unless you use Cloudflare (or similar), you have to wait 24-48 hours when making changes to your DNS records (e.g. register new domain, new subdomain, change IP, ...) to be sure everybody sees the new/"correct" version.

[11] Multiple locations advertise same IP! Routes take client to the closest one.





- Reduce content size for transfer
- Change HTTP to make better use of available bandwidth
- Change HTTP to avoid repeated transfers of the same content (→ caching, and proxies)
- Move content closer to client (→ CDNs)

A simple way to reduce the PLT are parallel connection since it doesn't require any changes on the server-side and there's plenty of bandwidth. Parallel connections however compete with each other for network resources. Persistent connections can be used as an alternative by making 1 TCP connection to 1 server and use it for multiple HTTP requests.

Other improvements include mod_pagespeed (Apache-specific; server rewrites/"compiles" pages (minify, compress ...)) or ~~SPDY~~ HTTP/2:

- Multiplexed (parallel) HTTP requests on one TCP connection
- Client priorities for parallel requests
- Compressed HTTP headers
- Server push of resources

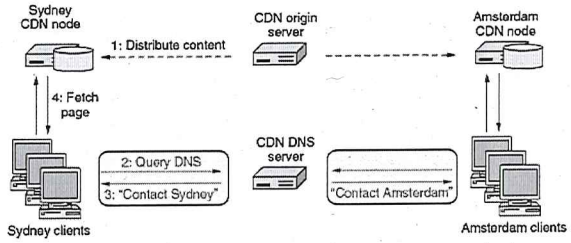
### Web proxies and caching

Users often revisit web pages and there's a big win from reusing local copy. The system *locally determines* if copy is still valid based on expiry information such as "Expires" header from server or by using a heuristic to guess (cacheable, freshly valid, not modified recently). In that case, content is then available right away. Or the system *revalidates copy with remote server* based on timestamp of copy such as "Last-Modified" header from server or based on content of copy such as "ETag" (unique identifier, computed by the server) header from server. Content is available after 1 RTT.

**Web proxies** are set up as an intermediary between a pool of clients and external web servers. Benefits for clients include greater caching and security checking and enables application of organizational access policies. When using proxy caching, clients benefit from larger, shared cache but benefits are limited by secure / dynamic content, as well as "long tail".

### Content Distribution Networks (CDN)

CDNs allow for efficient distribution of popular content and faster delivery for clients. They reduce server and network load while also decreasing the PLT. To place replicas close to the client, clever usage of DNS is needed.




Placing site replica at an ISP is win-win: it improves site experience and reduces bandwidth usage of the ISP.

### Peer-to-peer (P2P) / BitTorrent

Peer-to-peer content delivery runs without dedicated infrastructure and BitTorrent is one example.[12] While CDNs are efficient and reliable, they need dedicated infrastructure and centralized control. The goal of P2P is delivery without dedicated infrastructure or centralized control. This is still efficient at scale, and reliable. The key idea is to have participants (or peers) help themselves.

There are no servers on which to rely thus all communication must be peer-to-peer and self-organizing, not client-server.

| CHALLENGE | SOLUTION |
|---|---|
| Limited capabilities: how can one peer deliver content to all other peers? | Peer can send content to all other peers using a distribution tree<br>- Typically done with replicas over time<br>- Self-scaling capacity |
| Participation incentive: why will peers help each other? | A peer play 2 roles: download to help themselves, and upload to help others Coupling these two roles ("I'll upload for you if you upload for me") encourages cooperation. |
| Decentralization: how will peers find content? | Peers learn where to get content from by using DHTs (Distributed Hash Tables)<br>DHTs are fully-decentralized, efficient algorithms for a distributed index<br>- Index is spread across all peers<br>- Index lists peers to contact for content<br>- Any peer can lookup the index |

DHT

BitTorrent uses **torrents** which transfers files in pieces for parallelism. They are notable for treatment of incentives and uses either a tracker or decentralized index (DHT).

Steps to download a torrent:

1. Start with torrent description
2. Contact tracker to join and get list of peers (with at least seed peer)
2. Or, use DHT index for peers
3. Trade pieces with different peers
4. Favor peers that upload to you rapidly; "choke" peers that don't by slowing your upload to them

## Sources

Unless otherwise noted: Lecture slides presented by Adrian Perrig, available on the course website accompanying the course 252-0062-00L taught in the spring semester 2015 at ETH Zürich. Simple definitions might be from Wikipedia.

[12] Windows 10 will use P2P for Windows Updates.